The Pennsylvania State University
The Graduate School

USING ANTS TO FIND COMMUNITIES IN COMPLEX NETWORKS

A Thesis in
Computer Science
by
Mohammad Adi

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2014

The thesis of Mohammad Adi was reviewed and approved* by the following:

Thang N. Bui
Associate Professor of Computer Science
Chair, Mathematics and Computer Science Programs
Thesis Advisor

*Signatures are on file in the Gradute School.

Abstract

Many systems arising in different fields can be described as complex networks, a collection of nodes and edges. An interesting property of these networks is the presence of communities (or clusters), which represents a subset of nodes within the network such that the connections within these nodes are denser than the connections with the rest of the network. In this thesis, we give an ant-based algorithm for finding communities in complex networks. Ants are used to identify edges which are used to assign the nodes into different clusters. Tests on various synthetic and real-world networks show that the algorithm is able to extract the community structure very well and performs well against other algorithms.

Table of Contents

List of Figures vi

List of Tables vii

Acknowledgements viii

Chapter 1
Introduction 1

Chapter 2
Preliminaries 3
2.1 Problem Definition . . . 3
2.2 Previous Work . . . 4
2.2.1 Hierarchical Methods . . . 5
2.2.2 Modularity-based Methods . . . 5
2.2.3 Other Methods . . . 6
2.3 Ant Algorithms . . . 7

Chapter 3
Ant-Based Community Detection 10
3.1 Overview . . . 10
3.2 Initialization . . . 11
3.3 Exploration . . . 11
3.4 Construction . . . 15
3.5 Optimization Phase . . . 17
3.6 Parameters . . . 18

Chapter 4
Testing Algorithms 19
4.1 Synthetic Graphs . . . 19
4.1.1 Girvan-Newman Benchmark . . . 19
4.1.2 LFR Benchmark . . . 20
4.2 Partition Evaluation . . . 21
4.2.1 Normalized Mutual Information . . . 21
4.2.2 Modularity . . . 21

References **23**

List of Figures

3.1 Ant-Based Community Detection Algorithm 12
3.2 Exploration phase 13
3.3 Reset ants 16

List of Tables

3.1 Parameters in the algorithm 18

Acknowledgements

¡update later¿

Chapter 1

Introduction

Complex networks are extensively used to model various real-world systems such as social networks, technological (Internet and World Wide Web) networks, biological networks etc. These networks are modeled as graphs where nodes represent the objects in the system and edges represent the relationship among these objects. For example, in a social network, nodes can represent people and two nodes are connected by a link if they are friends with each other.

These networks exhibit distinctive statistical properties. The first property is the "small world effect", which implies that the average distance distance between vertices in a network is short [18]. The second is that the degree distributions follow a power-law [1], and the third one is network transitivity which is the property that two vertices who are both neighbors of the same third vertex, have an increased probabilibty of being neighbors of one another [37].

Another property which appears to be common to such networks is that of community structure (or clustering). While the concept of a community is not strictly defined in the literature as it can be affected by the application domain, one intuitive notion of a community is that it consists of a subset of nodes from the original graph which between them have a higher density of links as compared to their links with the rest of the graph. In this thesis, we describe an ant-based algorithm for automatically detecting communities in graphs, without specifying the number of communities beforehand.

Over the course of more than a decade, the task of finding communities in networks

has received enormous attention from researchers in different fields such as physics, statistics, computer science etc. As a result, there are currently a vast number of methods which can be used to evaluate the community structure of a network. These methods are described in the next chapter.

Ant algorithms have been previously used to detect communities in graphs [14] [32] [15]. In our approach, we use artificial ants which traverse the graph based solely on local information and deposit pheromone as they travel. This algorithm uses the cumulative pheromone on the edges to build up an initial clustering of the graph. Then a local optimization method is used to reassign the clusters of different nodes based on their degree distribution after which clusters are merged depending on certain rules to obtain the final partitioning of the graph.

The rest of this thesis is organized as follows. Chapter 2 provides more detailed information about the problem statement and covers the previous work done. The ant-based algorithm is described in Chapter 3. Chapter 4 covers metrics to evaluate partitions and Chapter 5 covers the performance of the algorithm on various synthetic and real-world graphs and compares it to existing algorithms. The conclusion is given in Chapter 6.

Chapter 2

Preliminaries

2.1 Problem Definition

Communities are generally defined to be subsets of vertices which have a high density of links within them. There are various possible definitions of a community and they are divided into mainly three classes: local, global and based on vertex similarity [11] [36]. A more general, quantitative criterion is described in [27] by considering the degree k_i of a node i belonging to a community $S \subset G$, where G is the graph representing the network. The degree of node i can be split as:

$$k_i(S) = k_i^{in}(S) + k_i^{out}(S) \tag{2.1}$$

where $k_i^{in}(S)$ is the number of connections to nodes in its subgraph S and $k_i^{out}(S)$ is the number of connections to nodes outside S. The authors define a community in 2 ways. The subgraph S is a community in the **strong sense** if:

$$k_i^{in}(S) > k_i^{out}(S), \forall i \in S \tag{2.2}$$

The subgraph S is a community in the **weak sense** if:

$$\sum_{i \in S} k_i^{in}(S) > \sum_{i \in S} k_i^{out}(S) \tag{2.3}$$

Even though networks can be directed, undirected, weighted or directed and weighted, we consider only undirected and unweighted networks. The problem of community detection can be defined as follows:

Input: An undirected, unweighted graph $G = (V, E)$ where V represents a set of nodes or vertices and E represents a set of edges or links.
Output: A partition $C = \{C_1, \ldots, C_k\}$ of G into k communities where $C_i \cap C_j = \varnothing, i, j = 1, \ldots, k, i \neq j$ and $C_i \subset V, \forall i$.

It is worth mentioning that while communities can also be hierarchical in nature, small communities can be nested within larger ones, in this work we only concentrate on finding disjoint communities.

2.2 Previous Work

The seminal paper by Girvan and Newman [12], resulted in a lot of research into the area of community detection, especially by physicists. As a result, these days there is a wide variety of community detection algorithms from fields like physics, computer science, statistics etc. Covering all of them is beyond the scope of this work, for a more thorough review one can refer the survey by Fortunato [11].

The methods for detecting communities can be broadly classified into hierarchical methods, modularity-based methods and other optimization methods involving statistics or dynamic processes on the graph.

2.2.1 Hierarchical Methods

These type of methods can be further divided into 2 subtypes: divisive hierarchical methods and agglomerative hierarchical methods.

Divisive hierarchical methods start from the complete graph, detect edges that connect different communities based on a certain metric such as edge betweenness [12], and remove them. Examples of these approaches can be found in [12] [27] [20].

Agglomerative hierachical methods initially consider each node to be in its own community then and merge communities until the whole graph is obtained. Examples can be found in [21] [2] [5].

2.2.2 Modularity-based Methods

Modularity [20] is a metric introduced by Girvan and Newman to evaluate the partitioning of a graph. It is way to quantify the clustering we have obtained in order to determine how good it might be and is a widely adopted quality metric. The idea is that the edge density of the nodes in a cluster should be higher than the expected density of the subgraph whose nodes are connected at random, but with the same degree sequence. This model is called the *null model*.

Using an adjacency matrix representation for the graph, modularity is written as follows:

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j) \tag{2.4}$$

where A is the adjacency matrix of the graph G, m is the number of edges in the graph, $\frac{k_i k_j}{2m}$ is the expected number of edges between nodes i and j in the null model and δ is the *Kronecker* functions whose value is 1 if i and j are in the same community and 0 otherwise. Since nodes which do not belong in the same cluster don't contribute towards modularity, it can be rewritten as:

$$Q = \sum_{i=1}^{k} \left(\frac{e_i}{m} - \left(\frac{d_i}{2m} \right)^2 \right) \tag{2.5}$$

where k is the number of communities, e_i is the total number of internal links in cluster i and d_i is the sum of the total degrees of nodes in i. So the first term represents the fraction of the total edges that are in a community and the second term represents the expected value of the fraction of edges in the null model. Values of Q approaching 1 (which is the maximum), indicate strong community structure [20]. In practice, the value usually ranges from 0.3 - 0.7.

Under the assumption that high values of modularity indicate good partitions, the partition corresponding to its maximum value for a given graph should be the best partition. This is the reasoning employed by modularity-based methods which try to optimize Q to partition a graph. Modularity-based methods are also the most popular methods to be employed for community detection. However, Brandes et. al. showed that maximizing modularity is a NP-hard problem [3], as a result the true maximum of modularity cannot be found in polynomial time unless P = NP. However, there are several algorithms based on different heuristics which approximate the modularity maximimum in a fair amount of time.

The first algorithm to maximize modularity was introduced in [21]. It is an agglomerative clustering approach where vertices are merged based on the maximum increase in modularity. Several other greedy techniques have been developed, some of these can be found in [2] [5] [19] [26] . A simulated annealing approaching to maximizing modularity is described in [13] [17]. Extremal optimization for maximizing modularity was used by Duch and Arenas [10].

Genetic algorithms have also been used for maximizing modularity [34] [23] [22] [24].

2.2.3 Other Methods

Various other techniques for community detection using methods based on statistical mechanics, information theory, random walks etc have been proposed .

Reichardt and Bornholdt [28] proposed a Potts model approach for community detection. Another algorithm based on the Potts model approach is described in [30], this method is fast and it's complexity is a little superlinear in the number of edges in the graph.

Random walks have also been used to detect communities. The motivation behind this is the idea that a random walker will spend a longer amount of time inside a community due to the high density of links within it. These methods are described in [38] [25] [35].

Information theoretic approaches use the idea of describing a graph by using less information than that encoded in its adjacency matrix. The aim is to compress the the amount of information required to describe the flow of information across the graph. Random walk is used as a proxy for information flow. The minimum description length (MDL) principle [29] can be used as a solution to this problem. The most notable algorithm using this principle, referred to as InfoMap, is described in [31].

2.3 Ant Algorithms

Before describing our algorithm, a review of ant algorithms is given. Ant algorithms are a probabilistic technique for solving computational problems using artificial ants. The ants mimic the behavior of an ant colony in nature for foraging food. As they travel, ants lay down a chemical trail called pheromone, which evaporates over time. The higher the pheromone on a path, the more likely it is to be chosen by the next ant that comes along.

Consider for example a food source and 2 possible paths to reach it, one shorter than the other. Assume two ants set off on both the paths simultaneously. The ant taking the shorter path will return earlier than the other one. Now this ant has covered the trip both ways while the other ant has not yet returned, so the concentration of pheromone on the shorter trail will be more. As a result, the next ant will be more likely to choose the shorter path due to its higher concentration of pheromone. This

leads to a further increase of pheromone on that path and eventually all ants will end up taking the shorter path.

Thus, ants can be used for finding good paths within a graph. It is this basic idea that is used in ant algorithms for solving computational problems, but there are different variations. The first such approach, called Ant System (AS), was applied to the Traveling Salesman Problem by Marc Dorigo [9]. In this approach, each ant is used to construct a tour and the pheromone level on all the edges in that tour is updated based on its length. Each ant picks the next destination based on its distance and the pheromone level on that link. A global update is applied everytime which evaporates the pheromone on all edges so the current best edges would be more like to be chosen by the next ants.

Since in AS each ant updates the pheromone globally, the run time can be quite high. Ant Colony System (ACS) was introduced to address this problem [8]. In ACS, a fixed number of ants are positioned on different cities and each ant constructs a tour, only the iteration best ant, the one with the shortest tour is used to update the pheromone. Ants also employ a local pheromone update in which the pheromone of an edge was reduced as an ant traversed it in order to encourage exploration.

Another variation of AS, the Max-Min Ant System (MMAS), was introduced by Stutzle and Hoos [33]. The first change in this model is that the pheromone values are limited to the interval $[\tau_{min}, \tau_{max}]$. Secondly, the global update for each iteration is either done by the iteration best ant or the ant which has the best solution from the beginning. This is to used so as to avoid early convergence of the algorithm. Additionally, the pheromone on each edge is initialized to τ_{max} so as to encourage exploration in the beginning of the algorithm. Apart from this, MMAS used the same structure of AS for edge selection and lack of local pheromone update. Both these variations were an improvement over the original AS.

The algorithm proposed here does not fall into the above class of ant algorithms, also called Ant Colony Optimization (ACO) algorithms, instead it falls into the category of Ant-Based Optimization (ABO) methods. While in ACO, ants build complete solutions to the problem, in this approach ants are only used to identify good regions

of the search space after which local search or construction methods are used to build the final solution. In ABO, ants only need local information as they traverse the graph. Choosing the next edge involves the pheromone value and some heurisitic information based on the rules specified for the ants. To the best of our knowledge, our algorithm is the first ABO method for detecting communities in complex networks.

Chapter 3

Ant-Based Community Detection

In this chapter, we describe our ant-based approach for detecting communities in complex networks. The algorithm is divided into three main phases: initialization, exploration and construction. The construction phase is followed by a local optimization step and a final merging step before obtaining the final partition.

3.1 Overview

Since our algorithm is ant-based, it consists of artificial ants which explore the graph based on a set of rules. In each cycle, all of the ants explore a local section of the graph by choosing edges based on their pheromone level. Edges with a higher pheromone value are more likely to be chosen and the pheromone on an edge is increased when an ant travels across it so that it is more likely to be chosen by other ants that follow. Based on the rules specified for exploration, ants try to discover intracommunity edges in their local section of the graph and as a result we expect them to have a higher value of pheromone towards the end of the exploration phase. The basic outline of the algorithm for the exploration phase is derived from [4].

Once the exploration phase is complete, we examine the edges with the highest pheromone levels and use them to build an intial clustering of the graph. After this, a local optimization step is used to boost the solution obtained based on the strong definition of a community as described in the previous chapter. Once this is done, clusters are merged depending upon certain rules which satisfy the weak definition of

a community. An outline of the algorithm is shown in Figure 3.1.

3.2 Initialization

In the initilization phase an ant is placed on each vertex of the graph. Next, we initialize all the edges with the same pheromone level, set to 1. This is done so that in the beginning of the algorithm all the edges have a chance to be selected, so it encourages exploration.

Each ant also maintains a tabu-list, initialized to null, which is a fixed length circular queue that stores a list of the previously visited vertices by an ant. If an ant chooses a vertex which is already present in the tabu-list, then the vertex is ignored and it attempts to choose another vertex. Tabu-list further encourages exploration as ants will not just tend to stay within a small vicinity of their initial location.

Each vertex is augmented with information about the number of nodes it has in common with each of its neighbors. Since the graph is stored in an adjacency list format, each vertex maintains a list of its neighbors. Calculating the number of nodes in common between two vertices can now be done by determining the number of common elements in their adjacency lists, or a set intersection operation.

3.3 Exploration

In this phase, the ants travel across the edges in the graph and lay pheromone along them. The aim of the exploration phase is to discover intracommunity edges which are used to build an initial clustering of the graph. Since communities can be intuitively thought of subgroups of vertices with a high density of internal links. This means that two adjacent vertices are more likely to be in the same community if the number of neighbors they have in common is large. This is the rule that the ants follow to discover intracommunity edges. The exploration phase is shown in Figure 3.3.

In each iteration, all the ants are moved in parallel on the graph for a fixed number of steps. This is continued until the maximum number of iterations is exceeded. For

```
AntCommunity(G)
Input: G = (V, E), graph whose community structure is to be found
Output: Set of k communities C = {C_1, ..., C_k}
begin
    //Initialization
    i ← 1
    foreach v ∈ V do
        Calculate the number of common nodes with all neighbors
    end
    Create the set of ants A, of size |V|
    InitAnts(G, A)

    //Exploration phase
    while i < i_max do
        AntsMove(G, A)
        Update phermone evaporation factor η
        ResetAnts(G, A)
        i ← i + 1
    end

    //Construction phase
    Sort E in decreasing order of pheromone
    PartitionOneLevel(G)
    ReassignClusters()
end

InitAnts(G, A)
begin
    for i = 0; i < |V|; i++ do
        A[i].location ← V[i]
        A[i].tabuList ← ∅
    end
    foreach e ∈ E do
        e.initPhm = 1
        e.nVisited = 0
    end
end
```

Figure 3.1: Ant-Based Community Detection Algorithm

```
AntsMove(G, A)
begin
    for s = 1 to maxSteps do
        if s mod updatePeriod == 0 then
            UpdatePheromone(G)
        end
        foreach a ∈ A do
            nAttempts ← 0
            moved ← False
            while not moved and nAttempts < 5 do
                v1 ← a.location
                Select an edge (v1, v2) at random and proportional to the
                phermone level and the size of their neighborhood overlap
                if v2 ∉ a.tabuList then
                    add v2 to a.tabuList
                    a.location ← v2
                    (v1, v2).nVisited++
                    moved ← True
                else
                    nAttempts++
                end
            end
        end
    end
end

UpdatePheromone(G) begin
    foreach e ∈ E do
        e.phm ← (1 − η) × e.phm + e.nVisited × e.initPhm
        e.nVisited ← 0
    end
    if e.phm < minPhm then
        e.phm = minPhm
    end
end
```

Figure 3.2: Exploration phase

purposes of efficiency, it is better to update the pheromone after a fixed number of steps instead of updating it after every step.

As mentioned previously, the movement of the ants is determined by the pheromone level of the edges incident to the current vertex and the number of nodes the current vertex has in common with its neighbors (neighborhood overlap). This way intra-community edges will have a higher probability of being selected leading to a higher level of pheromone.

Each edge is augmented with information about its pheromone level and the number of times it has been traversed. When an ant traverses an edge, it is scheduled to be increased by an amount equal to its initial pheromone value [4]. An edge may be selected by an ant with a probability that is proportional to the pheromone level of that edge and the number of nodes it has in common with the ants current vertex. Since the pheromone on all edges is initialized to 1, when the pheromone update is performed, the pheromone on each edge is increased by the number of times it was traversed.

Proportional selection is performed as follows. Each ant calculates the sum of the pheromone and number of common nodes it has with each neighbor and stores it in an array whose size is equal to the degree of the current vertex. While filling up the array we also maintain a running sum of the total, let this be denoted by sum. Then a random number between $[0, sum]$ is generated after which we sum up the array elements until the random value is reached. The index of this element corresponds to the index of the neighbor in the adjacency list and the ant chooses to move to that vertex. This way the ants will have a higher probability of choosing a vertex which belongs to the same community. When ant traverses an edge, it adds the next vertex to its tabu list to avoid visiting it multiple times.

We employ a couple of mechanisms to avoid getting caught in local optima. As mentioned previously, each ant maintains a tabu list of a fixed size and are prohibited from choosing vertices already present in the list. Also, the pheromone level of each edge is evaporated periodically by a certain factor which reduces over time. In the beginning, the evaporation rate is higher so as to encourage exploration and it is

gradually decreased. The minimum pheromone level is set to 1 as due to evaporation we don't want the pheromone level of an edge to become so low that it may never be considered again, this also avoids the ants from converging to a small set of edges.

The pheromone update is performed periodically, for reasons of efficiency. As mentioned previously, each edge keeps a track of the number of times it has been traversed since the previous update. This information is used along with the current pheromone level for the update. The formula is that mentioned in [4]:

$$e.phm = (1 - \eta)e.phm + e.nVisited \times e.initPhm \tag{3.1}$$

where e is the edge in the graph being updated, phm is the pheromone level of e, η is the evaporation rate, $nVisited$ is the number of times the edge was traversed since the last update cycle and $initPhm$ is the initial pheromone level, in this case $initPhm = 1, \forall e$.

At the end of each cycle, two operations are performed. First, the pheromone evaporation rate is updated by a constant factor. This is to aid exploration in the beginning by having a large evaporation rate and reducing it so that the ants begin to converge on a set of edges. The initial value of η is set to 0.5 and it is multiplied by 0.95 at the end of every iteration. Second, the ants are reset before the start of the next iteration. About half the ants stay in their current location while the other half are randomly placed as shown in Figure 3.3 The maximum number of iterations is fixed to 75.

3.4 Construction

At the end of the exploration phase we expect the intracommunity edges to have a higher pheromone level, the construction phase utilizes this information to cluster the graph. The output of the construction phase is a new graph called the weighted graph. Each community corresponds to a vertex of this graph. So each vertex can

```
ResetAnts(G, A)
begin
    foreach a ∈ A do
        if Random(0, 1) < 0.5 then
            a.location ← Random(0, |V| − 1)
        end
        a.tabuList ← ∅
    end
end
```

Figure 3.3: Reset ants

be thought of a supervertex which consists of vertices agglomerated from the original graph to form different comunities. The edges in this graph will correspond to the intercommunity edges. Even though there might be multiple such edges between two given communities in the original graph, only one such edge is necessary for representing the same in the weighted graph.

First, we sort the edges in decreasing order of pheromone. An array called $nodeToComm$ (of size $|V|$) stores the community membership of each node. It is initialized to -1 implying no node has been assigned to a cluster. The variable $comm$, initialized to 0, keeps track of the number of communities so far.

To build the weighted graph we read in each edge (i, j) sequentially. If $nodeToComm[i]$ and $nodeToComm[j]$ are both -1 then we create a new vertex representing a new cluster and add i, j to it. $nodeToComm[i]$ and $nodeToComm[j]$ are both set to $comm$ whose value is then incremented by 1. If i or j are assigned a community while the other isn't, then the node which isn't assigned a community is added to the other's cluster. If both i and j are in separate communities, ie $nodeToComm[i] \neq nodeToComm[j]$, then we create the edge $(nodeToComm[i], nodeToComm[j])$ in the weighted graph if it doesn't exist and update the number of crossing (intercommunity) edges in the new graph.

Additional information is stored in the weighted graph. For each vertex (or cluster) we keep a track of the original vertices of the graph which are a part of that cluster. Each node (of the original graph) stores the number of internal and external

links (and to which cluster) it has. Each cluster also stores the total pheromone (also called its *weight*) amongst all its internal links and the total pheromone (internal + pheromone along all its outgoing edges).

At the end of the construction phase we have built an initial clustering of the graph using the process described above. This weighted graph is now optimized by reassigning nodes which may have been correctly placed in the wrong cluster and then merging clusters to form more cohesive communities.

3.5 Optimization Phase

Based on the strong definition of a community as mentioned in chapter 2, we expect each node to have more connections to nodes within its own cluster than outside. We use this idea to reassign nodes which may have been incorrectly clustered.

Each node from the original graph, in the weighted graph, stores the number of links it has to its own cluster and the number of links to outside clusters. The nodes are sorted in decreasing order of outgoing links. Each node is examined to see if there is a cluster to which it has more links than its current cluster, if this is true then the node is assigned to that cluster. After this process is complete we proceed to merge clusters to obtain the final partition.

Since the ants are used to discover intracommunity edges, we can use the pheromone level in the weighted graph to merge clusters. The communities in the weighted graph so far may not be cohesive. Since we expect a community to have high density of internal links as compared to external links, this means that if a vertex has a high *weight* as compared to its total pheromone, then its a good community. Otherwise, its possible that the pheromone along one of the intercommunity edges will be higher as compared to its *weight* and we can merge the two communities to obtain a better solution. Since intercommunity edges from the original graph are condensed to one edge in the weighted graph, the more intercommunity edges we have, the higher we expect the pheromone to be along the corresponding edge in the weighted graph.

Table 3.1: Parameters in the algorithm

Parameter	Value	Comments
i_{max}	75	Maximum number of iterations
$maxSteps$	$\frac{1}{3}\|V\|$ or 75	Maximum number of steps in each iteration
η	0.5	Pheromone evaporation rate
$\Delta\eta$	0.95	Pheromone update constant
$updatePeriod$	$maxSteps/3$	Number of cycles between pheromone update
$LIST\_SIZE$	2 or 5	Tabu list size

The fraction of the total pheromone along each outgoing edge for a vertex is calculated. Since the graph is undirected, the fraction is calculated using the total contribution from both endpoints. The fractional values for each edge are stored in an array and sorted in decreasing order. For an edge with a high fractional value, it means that the two clusters have a high number of connections between them and a better solution would be obtained by merging them.

In the last step, the edges sorted in decreasing order of fractional values are used to merge different clusters. For each edge (u, v), if the fraction of the total pheromone on (u, v) is more than the fraction of the total pheromone in vertex u or v, they are merged. Else if more than half of the total pheromone lies within the both vertices u and v, then we don't merge them.

3.6 Parameters

The various parameters in the algorithm are mentioned in Table 3.1. These parameters aren't for a single type of graph but have been used for all graphs on which the algorithm is tested.

Chapter 4

Testing Algorithms

Since clustering algorithms always produce some partition of the graph it becomes necessary to evaluate how good a partition has been obtained. Testing an algorithm implies running it on a set of problem instances whose solution is already known and comparing it with the output of the algorithm. In the problem of community detection, while the intuitive definition is the same there are a wide variety of approaches to solving it. Due to this it is necessary to be able to use benchmark graphs whose community structure is known to be able to evaluate an algorithm. While this refers to computer generated graphs, in the literature real-world networks are also used. However it is necessary to note that the communities discovered by different algorithms may not be consistent due to the large variety of different implementations. In this case it is necessary to have a different metric to evaluate the partition.

This chapter discusses the two most widely used benchmark graphs for generating synthetic networks with known community structure and methods to evaluate the partition obtained on these graphs and also evaluate the partition obtained in the case of real world networks.

4.1 Synthetic Graphs

4.1.1 Girvan-Newman Benchmark

For computer generated graphs, a special class of graphs generated using the planted ℓ-partition model [6] is quite commonly used. This model partitions a graph with

$n = g\dot{\ell}$ vertices in ℓ groups with g vertices each. Each vertex has a fraction μ of its links to vertices outside its group and a fraction $1 - \mu$ to vertices in its group. If $1 - \mu > 0.5$ the density of intracluster edges is more and community structure exists.

The Girvan-Newman benchmark [12] is a special case of the planted ℓ-partition model where $n = 128$, $\ell = 4$ and the average degree of each vertex is 16. This benchmark quickly became a standard for testing algorithms. One would intuitively expect the graph to have community structure upto $\mu < 0.5$ and most algorithms begin to fail around that value.

4.1.2 LFR Benchmark

In the GN benchmark, while the graph is expected to have community structure upto $\mu < 0.5$ in principle communities exist up until $\mu = 0.75$ [16]. However, as mentioned previously most algorithms begin to fail around $\mu = 0.5$ which might be due to the graph being similar to a random graph as a result of changes in link distribution. Another drawback of the GN benchmark is that it doesn't take into account the structure of complex networks where node degree and community sizes follow a power law.

The LFR benchmark [16] was an improvement over the GN benchmark as it takes into account the structure of complex networks. It is also a special case of the planted ℓ-partition model where group sizes and node degrees vary according to a power law. This poses a much harder test to community detection algorithms. In addition, these graphs can be generated quickly, the complexity of the method mentioned in [16] is $O(m)$, where m is the number of edges in the graph.

We can generate different instances by varying μ to make the communities fuzzy and harder to detect. In the next section we describe metrics to evaluate the partition obtained by community detection algorithms for synthetic graphs and real world networks.

4.2 Partition Evaluation

4.2.1 Normalized Mutual Information

For synthetic graphs, the most widely adopted quality metric is the Normalized Mutual Information (NMI), as described in [7]. Here we define a *confusion matrix* $\mathbf{N}$, where the rows correspond to the "real" communities and the columns correspond to the communities found by an algorithm. N_{ij} represents the number of nodes in the real community i that appear in the found community j. The NMI, based on information theory is defined as follows:

$$I(A,B) = \frac{-2\sum_{i=1}^{c_A}\sum_{j=1}^{c_B} N_{ij}\log\left(\frac{N_{ij}N}{N_{i.}N_{.j}}\right)}{\sum_{i=1}^{c_A} N_{i.}\log\left(\frac{N_{i.}}{N}\right) + \sum_{j=1}^{c_B} N_{.j}\log\left(\frac{N_{.j}}{N}\right)} \tag{4.1}$$

where the number of real communities is denoted c_A and the number of found communities is denoted c_B, the sum over row i of matrix N_{ij} is denoted $N_{i.}$ and the sum over column j is denoted $N_{.j}$.

If the found partition is identical to the real one then $I(A,B) = 1$, which is its maximum value. If the partition found is totally independent of the real one then $I(A,B) = 0$.

4.2.2 Modularity

As described in Chapter 2, modularity is a widely adopted quality metric for evaluating partitions obtained on real world networks. The previous approach is not possible to apply here as we don't have prior information about the real community structure in such networks.

Modularity is calculated by computing the fraction of links that fall within a community as compared to the fraction if its nodes were connected randomly but keeping the same degree sequence. It was assumed that high modularity partitions corre-

spond to a good clustering which is the motivation behind modularity optimizing algorithms.

References

[1] Albert-Laszlo Barabasi and Reka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[2] V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E.L.J.S. Mech. Fast unfolding of communities in large networks. *J. Stat. Mech*, page P10008, 2008.

[3] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert G?rke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, 2008.

[4] T.N. Bui, Xianghua Deng, and C.M. Zrncic. An improved ant-based algorithm for the degree-constrained minimum spanning tree problem. *Evolutionary Computation, IEEE Transactions on*, 16(2):266–278, 2012.

[5] Aaron Clauset, M. E. J. Newman, , and Cristopher Moore. Finding community structure in very large networks. *Physical Review E*, pages 1– 6, 2004.

[6] Anne Condon and Richard M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures & Algorithms*, 18(2):116–140, 2001.

[7] Leon Danon, Albert Daz-guilera, and Jordi Duch. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, page 09008, 2005.

[8] M. Dorigo and L.M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[9] Marco Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[10] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72:027104, 2005.

[11] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(35):75 – 174, 2010.

[12] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[13] Roger Guimerà, Marta Sales-Pardo, and Lu´ is A. Nunes Amaral. Modularity from fluctuations in random graphs and complex networks. *Phys. Rev. E*, 70:025101, Aug 2004.

[14] Dongxiao He, Jie Liu, Bo Yang, Yuxiao Huang, Dayou Liu, and Di Jin. An ant-based algorithm with local optimization for community detection in large-scale networks. *CoRR*, abs/1303.4711, 2013.

[15] Di Jin, Dayou Liu, Bo Yang, Carlos Baquero, and Dongxiao He. Ant colony optimization with markov random walk for community detection in graphs. In *Proceedings of the 15th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*, PAKDD'11, pages 123–134, Berlin, Heidelberg, 2011. Springer-Verlag.

[16] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Phys. Rev. E*, 78:046110, Oct 2008.

[17] Claire P. Massen and Jonathan P. K. Doye. Identifying communities within energy landscapes. *Phys. Rev. E*, 71:046101, Apr 2005.

[18] Stanley Milgram. The Small World Problem. *Psychology Today*, 2:60–67, 1967.

[19] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[20] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, February 2004.

[21] M.E.J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69, September 2003.

[22] C. Pizzuti. A multiobjective genetic algorithm to find communities in complex networks. *Evolutionary Computation, IEEE Transactions on*, 16(3):418–430, 2012.

[23] Clara Pizzuti. Ga-net: A genetic algorithm for community detection in social networks. In Gnter Rudolph, Thomas Jansen, Simon Lucas, Carlo Poloni, and Nicola Beume, editors, *Parallel Problem Solving from Nature PPSN X*, volume 5199 of *Lecture Notes in Computer Science*, pages 1081–1090. Springer Berlin Heidelberg, 2008.

[24] Clara Pizzuti. Boosting the detection of modular community structure with genetic algorithms and local search. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 226–231, New York, NY, USA, 2012. ACM.

[25] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. In pInar Yolum, Tunga Gngr, Fikret Grgen, and Can zturan, editors, *Computer and Information Sciences - ISCIS 2005*, volume 3733 of

Lecture Notes in Computer Science, pages 284–293. Springer Berlin Heidelberg, 2005.

[26] Josep M. Pujol, Javier Béjar, and Jordi Delgado. Clustering algorithm for determining community structure in large networks. *Phys. Rev. E*, 74:016107, Jul 2006.

[27] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9):2658–2663, 2004.

[28] Jörg Reichardt and Stefan Bornholdt. Detecting fuzzy community structures in complex networks with a potts model. *Phys. Rev. Lett.*, 93:218701, Nov 2004.

[29] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465 – 471, 1978.

[30] Peter Ronhovde and Zohar Nussinov. Local resolution-limit-free potts model for community detection. *Phys. Rev. E*, 81:046114, Apr 2010.

[31] Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

[32] S. Sadi, S. Oguducu, and A.S. Uyar. An efficient community detection method using parallel clique-finding ants. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–7, 2010.

[33] Thomas Sttzle and Holger H. Hoos. Maxmin ant system. *Future Generation Computer Systems*, 16(8):889 – 914, 2000.

[34] Mursel Tasgin and Haluk Bingol. Community detection in complex networks using genetic algorithm. In *ECCS '06: Proc. of the European Conference on Complex Systems*, April 2006.

[35] Stijn van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.

[36] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[37] D.J. Watts and S.H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, (393):440–442, 1998.

[38] Haijun Zhou. Network landscape from a brownian particle's perspective. *Phys. Rev. E*, 67:041908, Apr 2003.